ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΚΡΟΙΤΟΡ ΚΑΤΑΡΤΖΙΟΥ ΙΩΑΝ Π21077

ΕΡΓΑΣΙΑ ΜΑΘΗΜΑΤΟΣ ΣΥΣΤΗΜΑΤΑ ΠΟΛΥΜΕΣΩΝ

ΠΕΙΡΑΙΑΣ

Ιούνιος 2024

ΠΡΟΛΟΓΟΣ

Η παρούσα εργασία αναπτύχθηκε ως μέρος του μαθήματος Συστήματα Πολυμέσων, με κύριο στόχο τη δημιουργία προγραμμάτων για εξοικείωση με τη συμπίεση και κωδικοποίηση video.

ΕΚΦΩΝΗΣΗ

Θέμα (30%του τελικού βαθμού): Έστω ασυμπίεστο video της επιλογής σας, διάρκειας 5 s – 15 s. Υποθέστε ότι ανά 12 πλαίσια το πρώτο είναι πάντα τύπου I και τα υπόλοιπα τύπου P.

I. Κάθε πλαίσιο P προβλέπεται χωρίς αντιστάθμιση κίνησης από το προηγούμενο πλαίσιο. Υπολογίστε και απεικονίστε την ακολουθία εικόνων σφάλματος και κωδικοποιήστε την χωρίς απώλειες. Υλοποιήστε τον κωδικοποιητή/αποκωδικοποιητή.

II. ii) Υλοποιήστε την τεχνική εξαντλητικής αντιστάθμισης κίνησης για την συμπίεση της ακολουθίας πλαισίων χρησιμοποιώντας αντιστάθμιση κίνησης σε macroblocks μεγέθους 16x16, ακτίνα αναζήτησης k=8 και τεχνική σύγκρισης macroblocks της επιλογής σας.

III. iii) Να επιταχυνθεί η διαδικασία αντιστάθμισης κίνησης υλοποιώντας λογαριθμική αναζήτηση. Υπολογίστε τα διανύσματα κίνησης και απεικονίστε την ακολουθία εικόνων πρόβλεψης και εικόνων σφαλμάτων. Υλοποιήστε τον κωδικοποιητή/αποκωδικοποιητή.

IV. iv) Υπολογίστε το βαθμό συμπίεσης για τις περιπτώσεις i) και iii)

- Μπορείτε να χρησιμοποιήσετε τη βιβλιοθήκη opencv μόνο για ανάγνωση/αποθήκευση αρχείων.
- Δεν μπορείτε να χρησιμοποιήσετε έτοιμες συναρτήσεις για τον υπολογισμό των διανυσμάτων κίνησης.
- Η αντιγραφή οδηγεί σε μηδενισμό όλων των εμπλεκόμενων εργασιών. Ομοίως, μηδενίζονται οι εργασίες που χρησιμοποιούν bots.

ΠΕΡΙΕΧΟΜΕΝΑ

1. ΕΙΣΑΓΩΓΗ

1.1 ΣΤΟΧΟΙ ΕΡΓΑΣΙΑΣ

Βασικός στόχος της εργασίας είναι η υλοποίηση αλγοριθμικών προγραμμάτων για τους διάφορους τρόπους διαχείρισης της συμπίεσης του video.

2. ΠΕΡΙΓΡΑΦΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

2.1 ΠΑΡΟΥΣΙΑΣΗ ΑΡΧΙΚΗΣ ΣΚΕΨΗΣ

2.1.1 ΕΡΩΤΗΜΑ i

Το πρώτο ερώτημα του προβλήματος αφορούσε τον υπολογισμό και την απεικόνιση των εικόνων σφαλμάτων που προκύπτουν από την συμπίεση στην οποία κάθε πλαίσιο προβλέπεται χωρίς αντιστάθμιση κίνησης. Έτσι, λοιπόν, θα πρέπει να υπολογίζουμε κάθε πλαίσιο P (P-frame) με βάση το προηγούμενο, άρα, για κάθε πλαίσιο P θα πρέπει να υπολογίζουμε την διαφορά P - (P-1), όπου P το τρέχον πλαίσιο και P-1 το προηγούμενο πλαίσιο. Ύστερα κωδικοποιούμε με Huffman κωδικοποίηση το πλαίσιο διαφορών και το υπερθέτουμε στην ακολουθία εξόδου. Όσον αφορά τα πλαίσια I (I-frames), αυτά θα κωδικοποιούνται ανεξάρτητα από τα υπόλοιπα δηλαδή χωρίς να ληφθεί υπόψιν κάποιο προηγούμενο πλαίσιο, οπότε εφαρμόζεται απευθείας Huffman. Έτσι θα διασφαλιστεί ότι τα πλαίσια I μπορούν να αποκωδικοποιηθούν ανεξάρτητα, ενώ τα πλαίσια P βασίζονται στο προηγούμενο πλαίσιο για την ανακατασκευή τους. Ύστερα, θα αποθηκετούν σε byte μορφή τα κωδικοποιημένα πλαίσια, σε ένα binary αρχείο (.bin)

Ομοίως στην αποκωδικοποίηση θα εφαρμοστεί η αντίστροφη διαδικασία, όπου πρώτα μετατρέπονται τα binary αρχεία από byte σε int, και τα πλαίσια τα πλαίσια P, αφότου αποκωδικοποιηθούν τα πλαίσια διαφορών, θα υπολογιστούν ως (P-1) + difference, όπου difference είναι η διαφορά του P-1 από το P, ενώ στη συνέχεια τα υπερθέτουμε στην έξοδο. Όσον αφορά τα πλαίσια I, αυτά θα αποκωδικοποιηθούν ανεξάρτητα και θα υπερτεθούν στην έξοδο.

***Η μέθοδος αυτή βασίστηκε στο σχήμα 8-6 της σελίδας 257 του συγγράματος και στις διαλέξεις.**

2.1.2 ΕΡΩΤΗΜΑ ii

Το δεύτερο ερώτημα του προβλήματος αφορούσε τον υπολογισμό των πλαισίων διαφορών και των διανυσμάτων κίνησης που προκύπτουν από την αντιστάθμιση κίνησης μέσω της εξαντλητικής αναζήτησης.. Δηλαδή για κάθε P πλαίσιο θα υπλογίζεται τα διανύσματα κίνησης αναζητώντας σε κάθε θέση γύρω από το μακρομπλοκ που βρίσκεται εντός του εύρους της ακτίνας αναζήτησης [-k, +k] το καλύτερο ταίριασμα σύμφωνα με την τεχνική σύγκρισης SSD (Sum of Squared Differences) και θα επιστρέφεται κάθε φορά το καλύτερο ταίριασμα, και τα αντίστοιχα διανύσματα διαφορών.

Ομοίως στην αποκωδικοποίηση θα εφαρμόζεται η αντίστροφη διαδικασία όπου θα εφαρμόζονται τα διανύσματα κίνησης για να ανακατασκευαστούν τα πλαίσια.

***Η μέθοδος αυτή βασίστηκε στο σχήμα 8-5 8-6, 8-7 και 8-10 των σελίδων 259, 261, 262 και 265 του συγγράματος και στις διαλέξεις.**

2.1.2 ΕΡΩΤΗΜΑ ii

Το δεύτερο ερώτημα του προβλήματος αφορούσε τον υπολογισμό των πλαισίων διαφορών και των διανυσμάτων κίνησης που προκύπτουν από την αντιστάθμιση κίνησης μέσω της εξαντλητικής αναζήτησης. Δηλαδή για κάθε P πλαίσιο θα υπολογίζεται τα διανύσματα κίνησης αναζητώντας σε κάθε θέση γύρω από το μακρομπλοκ που βρίσκεται εντός του εύρους των 8 περιοχών αναζήτησης [*(0, 0), (k, 0), (-k, 0), (0, k), (0, -k), (k, k), (k, -k), (-k, k), (-k, -k)*] το καλύτερο ταίριασμα σύμφωνα με την τεχνική σύγκρισης SSD (Sum of Squared Differences) και θα επιστρέφεται κάθε φορά το καλύτερο ταίριασμα, και τα αντίστοιχα διανύσματα διαφορών.

Ομοίως στην αποκωδικοποίηση θα εφαρμόζεται η αντίστροφη διαδικασία όπου θα εφαρμόζονται τα διανύσματα κίνησης για να ανακατασκευαστούν τα πλαίσια.

***Η μέθοδος αυτή βασίστηκε στο σχήμα 8-18 και 8-19 των σελίδων 273 και 274 του συγγράματος και στις διαλέξεις.**

2.1.2 ΕΡΩΤΗΜΑ iv

Θα υπολογιστούν οι λόγοι συμπίεσης σύμφωνα με τον τύπο:

(σύμβολα κανονικής ροής)/ (σύμβολα συμπιεσμένης ροής)

Δηλαδή στον αριθμητή θα τοποθετηθεί ο αρχικός αριθμός των bit του ασυμπίεστου βίντεο (binary αρχείο) και στον παρονομαστή ο αριθμός των bit του συμπιεσμένου βίντεο (binary αρχείο).

2.2 ΑΝΑΛΥΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

Για την ανάπτυξη των προγραμμάτων, χρησιμοποιήθηκε η Pyhton3. Αρχικά θα δοθούν κάποιες βασικές πληροφορίες και στη συνέχεια θα αναλυθούν τα κυρίως πρόγραμματα.

2.2.1 ΓΕΝΙΚΗ ΠΕΡΙΓΡΑΦΗ

Οι βασικές βιβλιοθήκες που χρησιμοποιήθηκαν είναι η OpenCV για ανάγνωση και αποθήκευση των βίντεο και η numpy για διευκόλυνση των αριθμητικών πράξεων. Επιπλέον για την Huffman κωδικοποίηση χρησιμοποιήθηκαν τρεις ακόμη βιβλιοθήκες που θα αναφερθούν παρακάτω.

2.3.1 ΑΝΑΛΥΣΗ ΒΑΣΙΚΩΝ ΠΡΟΓΡΑΜΜΑΤΩΝ

2.3.1.1 huffman_bitstream.py

Το αρχείο αυτό περιέχει τόσο τον κώδικα για την κωδικοποίηση όσο και για την αποκωδικοποίηση χωρίς απώλειες, μέσω Huffman.

class Node

Η κλάση αυτή αποτελεί τους κόμβους του Huffman δένδρου που θα δημιουργηθεί, για την υλοποίηση του οποίου θα χρησιμοποιηθεί δομή δεδομένων σωρού ελαχίστων (minHeap). Παρακάτω φαίνεται ο ψευδοκώδικας που αποτέλεσε την βάση για την υλοποίηση του (βλέπε πηγές).

1. Push all the characters in ch[] mapped to corresponding frequency freq[] in priority queue.
2. To create Huffman Tree, pop two nodes from priority queue.
3. Assign two popped node from priority queue as left and right child of new node.
4. Push the new node formed in priority queue.
5. Repeat all above steps until size of priority queue becomes 1.
6. Traverse the Huffman Tree (whose root is the only node left in the priority queue) to store the Huffman Code

Ο κάθε κόμβος αρχικοποιείται έχοντας τα δεδομένα (ο κώδικας huffman για τον συγκεκριμένο αριθμό), την συχνότητα και τα δύο παιδιά (αριστερό και δεξί). Επιπλέον ορίζουμε πως θα συγκρίνονται στιγμιότυπα της κλάσης Nodes, όταν χρησιμοποιείται ο τελεστής '<', δηλαδή θα συγκρίνεται το χαρακτηριστικό της συχνότητας των δύο κόμβων για να εξαχθεί κάποιο αποτέλεσμα.

def calcFreq(dataset)

Η συνάρτηση αυτή υπολογίζει τις συχνότητες των αριθμών στο σύνολο που λαμβάνεται ως όρισμα, χρησιμοποιώντας την συνάρτηση defaultdict() της βιβλιοθήκης collections και το λεξικό freq το οποίο ύστερα θα ενημερωθεί με τις νέες τιμές που αντιπροσωπεύουν την συχνότητα του κάθε αριθμού. Τέλος, επιστρέφει το λεξικό freq με τις συχνότητες.

def buildHuffmanTree(freq)

Η συνάρτηση αυτή δημιουργεί το δένδρο Huffman με κόμβους τα στοιχεία του λεξικού freq (data, freq), που λαμβάνεται ως όρισμα και το οποίο ύστερα γίνεται σωρός ελαχίστων (minHeap), όπου η μικρότερη συχνότητα θα βρίσκεται πάντα στην κορυφή, χρησιμοποιώντας την συνάρτηση heapify() της βιβλιοθήκης heapq. Στη συνέχεια, έως ότου το μήκος του σωρού είναι < 1 επαναλαμβάνεται η εξής διαδικασία:

1. Αφαιρούνται οι δύο κόμβοι με την μικρότερη συχνότητα (δεξί και αριστερό παιδί της ρίζας)
2. Αρχικοποιείται ένας νέο κόμβο χωρίς δεδομένα του οποίου η συχνότητα θα είναι το άθροισμα των συχνοτήτων των κόμβων που αφαιρέθηκαν και ορίζονται το αριστερό και το δεξί παιδί στον κόμβο αυτό.
3. Προσθέτεται ο κόμβος αυτός πίσω στον σωρό ελαχίστων (heappush)

Τέλος, η συνάρτηση αυτή επιστρέφει τη ρίζα του σωρού.

def generateCodes(root, current_code="")

Η αναδρομική συνάρτηση παράγει τους κωδικούς Huffman για το δένδρο που παρήγαγε η προηγούμενη συνάρτηση. Λαμβάνει ως όρισμα την ρίζα του σωρού ελαχίστων, που επιστρέφεται από την buildHuffmanTree(freq) και τον κωδικό που θα ανατεθεί. Αρχικά ελέγχει αν η ρίζα και το πεδίο data του κόμβου είναι μη κενά, και ύστερα αναθέτει ανδρομικά την τιμή '0' εάν είναι το αριστερό παιδί και την τιμή '1' εάν είναι το δεξιό παιδί. Οι τιμές αυτές αποθηκεύονται στο λεξικό codes.

def encode(dataset)

Η συνάρτηση αυτή εκτελεί την κωδικοποίηση του συνόλου δεδομένων, που λαμβάνεται ως όρισμα, με βάση τους κωδικούς που παρήχθησαν. Επιστρέφει την κωδικοποιημένη συμβολοσειρά, αντιπροσωπεύοντας το δοσμένο σύνολο.

def padEncodedString(encoded_string)

Η συνάρτηση αυτή λαμβάνει ως όρισμα την κωδικοποιημένη συμβολοσειρά και προσθέτει '0' ώστε να είναι πολλαπλάσιο του 8 αφού ύστερα θα γραφτεί σε ένα bitstream file ως δυαδική ροή (σε bytes). Ο αριθμός τον '0' που θα προστεθούν στο τέλος της ακολουθίας δίνεται από τον τύπο extra_padding = 8 - len(encoded_string) % 8. Στη συνέχεια δημιουργείται μία 8-bit αναπαράσταση του extra_padding ώστε στη συνέχεια να γνωρίζει η συνάρτηση αφαίρεσης του padding πόσα bits προστέθηκαν. Επιστρέφεται η κωδικοποιημένη συνάρτηση με τα επιπρόσθετα bits.

def getByteArray(padded_encoded_string)

Η συνάρτηση αυτή παίρνει ως όρισμα το padded_encoded_string και δημιουργεί ένα πίνακα byte στον οποίο προσθέτουμε σε μορφή byte κάθε 8άδα χαρακτήρων από την λαμβανόμενη συμβολοσειρά. Επιστρέφει το πίνακα byte.

def removePadding(padded_encoded_string)

Η συνάρτηση αυτή αφαιρεί τα επιπρόσθετα bit που προστέθηκαν ώστε να είναι δυνατή η μετατροπή της συμβολοσειράς σε δυαδική ροή καθώς λαμβάνει ως όρισμα την συμβολοσειρά στην οποία προστέθηκαν τα bit. Αρχικά προσδιορίζεται πόσα 0 προστέθηκαν από τα πρώτα 8 bit της συμβολοσειράς, και ύστερα αυτά αφαιρούνται ώστε να μετατραπεί η ροή byte πίσω σε 0 και 1. Επιστρέφεται η κωδικοποιημένη συμβολοσειρά.

def decode_file(codes, encoded_string)

Η συνάρτηση αυτή αποκωδικοποιεί την κωδικοποιημένη με Huffman συμβολοσειρά, λαμβάνοντας ωε όρισμα του κωδικούς Huffman που χρησιμοποιήθηκαν στην κωδικοποιητής για το κάθε σύμβολο και την κωδικοποιημένη συμβολοσειρά. Ύστερα για κάθε bit της συμβολοσειράς ελέγχεται αν αυτό αντιστοιχεί σε κάποιο κωδικό, και αν αυτό ισχύει τότε αυτός προστίθεται στη λίστα με τα αποκωδικοποιημένα δεδομένα. Σε κάθε επόμενη επανάληψη το επόμενο bit της συμβολοσειράς προστίθεται στο ήδη υπάρχον ώστε να καλυφθούν όλες οι περιπτώσεις (όλοι οι κωδικοί). Επιστρέφεται η αποκωδικοποιημένη συμβολοσειρά.

def writeHuffmanCodes(codes, file):

Η συνάρτηση είναι βοηθητική και λαμβάνει ως όρισμα τους κωδικούς Huffman που χρησιμοποιήθηκαν κατά την διάρκεια της κωδικοποίησης και το όνομα του αρχείου στο οποίο θα αποθηκευτεί το λεξικό με τους κωδικούς, το οποίο είναι της μορφής {key: value}.

Αυτό επιτυγχάνεται με τη χρήση της βιβλιοθήκης pickle η οποία είναι χρήσιμη για serialization και συγκεκριμένα με την dump().

def readHuffmanCodes(codes, file):
Ομοίως με την προαναφερόμενη συνάρτηση, αυτή διαβάζει από το pickle αρχείο το λεξικό με τους Huffman κωδικούς.

2.3.1.2 i_encoder.py

Το αρχείο αυτό περιέχει τον κωδικοποιητή για το ερώτημα i. Ακολουθώντας την σκέψη που παρουσιάστηκε παραπάνω και χρησιμοποιώντας την βιβλιοθήκη της OpenCV, γίνεται η ανάγνωση των frames από το επιλεγμένο video, το οποίο είναι ένα βίντεο ήπιας κίνησης, το οποίο απεικονίζει θαλασσινά σε μία λαϊκή αγορά της Ιαπωνίας. Το βίντεο έχει διάρκεια 13 δευτερόλεπτα, είναι RGB και σε διαστάσεις 720x1280. Πιο συγκεκριμένα , χρησιμοποιούνται οι συναρτήσεις *VideoCapture()* για ανάγνωση των frames και *VideoWrite()* για αποθήκευση των παραγόμενων frames στο output. Τα πλαίσια θα λαμβάνονται από το βίντεο μέσω ενός βρόχου ο οποίος θα τερματίζει μόνο όταν δεν έχουν μείνει άλλα πλαίσια προς επεξεργασία. Συνεχίζοντας, για να είμαστε σε θέση να συγκρίνουμε τους λόγους συμπίεσης για τα κωδικοποιημένα binary αρχεία, θα πρέπει να ελεγχθεί το μέγεθος του binary αρχείου αρχείου που θα πρόκυπτε εάν δεν εφαρμοζόταν η κωδικοποίηση Huffman. Άρα, εντός του βρόχου που λαμβάνονται τα frames από το βίντεο, εκτός από το να τα κωδικοποιούμε θα πρέπει πρώτα να αποθηκεύονται ως έχουν (raw format) στο αρχείο *uncompressed_frames_bin*. Μετά θα πρέπει να εξεταστεί πιο frame είναι $12^{ο}$ κάθε φορά, δηλαδή πολλαπλάσιο του 12. Αυτό γίνεται μέσω μιας συνθήκης που εξετάζει αν ο μετρητής frame_idx modulo 12 είναι ίσο ή διάφορο του 0. Ύστερα πραγματοποιείται η αφαίρεση μεταξύ του τρέχοντος πλαισίου και του προηγούμενου εάν το τρέχον είναι πλαίσιο P, ή συνεχίζεται στο επόμενο βήμα εάν το τρέχον πλαίσιο είναι I. Επιπροσθέτως, τα πλαίσια διαφορών γράφονται στην έξοδο (βίντεο) *differencies_i.avi* ενώ τα πλαίσια διαφορών και τα πλαίσια I γράφονται στην έξοδο (βίντεο) *encoded_i.avi.*
Ένα σημαντικό βήμα σε όλη την διαδικασία, αποτελεί η μετατροπή του προς κωδικοποίηση και αποθήκευση πλαισίου σε 1D array μήκους 720*1280*3=2764800, εφόσον προηγουμένως είναι 3D λόγω των τριών καναλιών χρώματος. Αυτό επιτυγχάνεται μέσω της *flat* η οποία επιστρέφει μία 1D αναπαράστασης του τρισδιάστατου πίνακα.
Μετά ξεκινάει η διαδικασία της κωδικοποίησης Huffman, χρησιμοποιώντας το *huffman_bitsream.py*, και αφού δημιουργηθεί η κωδικοποιημένη εκδοχή του πλαισίου, αυτό γράφεται σε ένα binary αρχείο, με όνομα *encoded_frames.bin* ως εξής: τα πρώτα 4 bytes πάντα υποδηλώνουν το μέγεθος (σε bytes) του πλαισίου που πρόκειται να γραφτεί ώστε ο αποκωδικοποιητής να διαβάζει αντίστοιχα τα πλαίσια, όλα από το ίδιο αρχείο, χωρίς να υπάρχουν συγκρούσεις. Συνεχίζοντας, αποθηκεύεται στην αντίστοιχη θέση του λεξικού codes (σύμφωνα με τον μετρητή frame_idx) οι κώδικες που χρησιμοποιήθηκαν για το αντίστοιχο πλαίσιο. Τέλος, το αρχείο με τους κώδικες αποθηκεύεται σε ένα .pkl αρχείο me όνομα *huffman_codes.pkl*.

2.3.1.3 i_decoder.py

Το αρχείο αυτό περιέχει τον αποκωδικοποιητή για το ερώτημα i. Ακολουθώντας την αντίστροφή διαδικασία αποκωδικοποιούνται τα πλαίσια και γράφονται στην έξοδο βίντεο *decoded_i.avi.* Συγκεκριμένα, εντός ενός βρόχου, διαβάζεται το αρχείο *encoded_frames.bin,*

πρώτα τα 4 bytes για να προσδιοριστεί το μήκος που θα πρέπει να διαβαστεί ύστερα, για να εξάγει το τρέχον κωδικοποιημένο frame. Εν συνεχεία, το εξαγόμενο κωδικοποιημένο πλαίσιο μετατρέπεται σε binary string, αφαιρείται το padding και αποκωδικοποιεί το πλαίσιο σύμφωνα με τους κώδικες Huffman του λεξικού codes της τρέχουσας θέσης, σύμφωνα με τον μετρητή frame_idx.
Ένα σημαντικό βήμα σε όλη την διαδικασία, αποτελεί η μετατροπή του αποκωδικοποιημένου πλαισίου σε 3D array, εφόσον προηγουμένως είναι 1D λόγω της μετατροπής που προηγήθηκε. Αυτό επιτυγχάνεται μέσω της *reshape(x,y,z)* η οποία επιστρέφει μία z-D αναπαράστασης του δοσμένου πίνακα. Επομένως μετατρέπουμε το πλαίσιο σε διαστάσεις 720x1280x3.
Τέλος εφόσον το πλαίσιο είναι I γράφεται απλώς στην έξοδο, ενώ εάν είναι P τότε προστίθεται στο προηγούμενο ώστε να ανακατασκευαστεί.

2.3.1.4 motion_compensation.py

Αυτός ο κώδικας περιέχει τις συναρτήσεις για την υλοποίηση της αντιστάθμισης κίνησης τόσο μέσω της εξαντλητικής αναζήτησης όσο και μέσω της λογαριθμικής αναζήτησης.

def exhaustiveSearch(search_radius_k, macroblock_size, this_block, reference_frame, x, y)
Η συνάρτηση αυτή πραγματοποιεί την εξαντλητική αναζήτηση στην αντιστάθμιση κίνησης, με ορίσματα την παράμετρο αναζήτησης k, το μέγεθος μακρομπλοκ, το τρέχον μπλοκ, το πλαίσιο αναφοράς και τα x, y που αναπαριστούν το μήκος και το πλάτος του πλαισίου. Αρχικά εντός ενός βρόχου, αναζητείται στο εύρος [-k, +k] το καλύτερο ταίριασμα μπλοκ μέσω της εξής διαδικασίας:

1. Σε κάθε προσπέλαση θεωρείται μία νέα θέση στην οποία το x, y μεταβάλλεται κατά dx, dy αντίστοιχα.
2. Συνεχίζοντας, γίνεται έλεγχος έτσι ώστε η μεταβολή του x και του y να βρίσκεται εντός των ορίων του τρέχοντος πλαισίου, τόσο κατά μήκος όσο και κατά ύψος.
3. Στο επόμενο βήμα εξάγεται το αντίστοιχο μακρομπλοκ από το πλαίσιο αναφοράς, δηλαδή από την θέση *(y+dy)* εώς *(y+dy) + macroblock_size* και *(x+dx)* εώς *(x+dx) + macroblock_size* αντίστοιχα.
4. Ύστερα υπολογίζεται το σφάλμα μέσω του SSD (Sum of Squared Differences) για την σύγκριση των μακρομπλοκ:

$$SS = \sum (x_i - \bar{x})^2$$

Εικόνα 4. SSD

5. Τέλος ελέγχεται αν το τρέχον σφάλμα είναι μικρότερο από το ήδη μικρότερο, και αν ισχύει η συνθήκη τότε αυτό αντικαθίσταται.

Η συνάρτηση επιστρέφει το μακρομπλοκ με το καλύτερο ταίριασμα (ελάχιστο σφάλμα) καθώς και τα διανύσματα κίνησης που οδήγησαν στην εύρεση του μακρομπλοκ αυτού.
(βάσει σελίδας 262 του συγγράματος και διαλέξεων)

def logarithmicSearch(search_radius_k, macroblock_size, this_block, reference_frame, x, y):
Η συνάρτηση αυτή πραγματοποιεί την λογαριθμική αναζήτηση στην αντιστάθμιση κίνησης, με ορίσματα την παράμετρο αναζήτησης k, το μέγεθος μακρομπλοκ, το τρέχον μπλοκ, το πλαίσιο αναφοράς και τα x, y που αναπαριστούν το μήκος και το πλάτος του πλαισίου. Αρχικά εντός ενός βρόχου που τερματίζει μόλις το βήμα (step) γίνει μικρότερο του 1, κάθε φορά αναζητείται σε 8 κατευθύνσεις (2 οριζόντια, 2 κάθετα, 2 διαγώνια) το καλύτερο ταίριασμα μπλοκ μέσω της εξής διαδικασίας:

1. Ορίζουμε τα εξής 8 σημεία [(0, 0), (step, 0), (-step, 0), (0, step), (0, -step), (step, step), (step, -step), (-step, step), (-step, -step)] και το αρχικό βήμα ίσο με k.
2. Σε κάθε προσπέλαση θεωρείται μία νέα θέση στην οποία το x, y μεταβάλλεται κατά dx, dy αντίστοιχα.
3. Συνεχίζοντας, γίνεται έλεγχος έτσι ώστε η μεταβολή του x και του y να βρίσκεται εντός των ορίων του τρέχοντος πλαισίου, τόσο κατά μήκος όσο και κατά ύψος.
4. Στο επόμενο βήμα εξάγεται το αντίστοιχο μακρομπλοκ από το πλαίσιο αναφοράς, δηλαδή από την θέση *(y+dy)* εώς *(y+dy) + macroblock_size* και *(x+dx)* εώς *(x+dx) + macroblock_size* αντίστοιχα.
5. Ύστερα υπολογίζεται το σφάλμα μέσω του SSD (Sum of Squared Differences) για την σύγκριση των μακρομπλοκ.
6. Ελέγχεται αν το τρέχον σφάλμα είναι μικρότερο από το ήδη μικρότερο, και αν ισχύει η συνθήκη τότε αυτό αντικαθίσταται.
7. Το βήμα γίνεται υποδιπλασιάζεται

Η συνάρτηση επιστρέφει το μακρομπλοκ με το καλύτερο ταίριασμα (ελάχιστο σφάλμα) καθώς και τα διανύσματα κίνησης που οδήγησαν στην εύρεση του μακρομπλοκ αυτού.
(βάσει σελίδας 273-274 του συγγράματος και διαλέξεων)

def motionCompensation(search_radius_k, macroblock_size, this_frame, reference_frame, search):
Η συνάρτηση αυτή πραγματοποιεί την αντιστάθμιση κίνησης, με ορίσματα την παράμετρο αναζήτησης k, το μέγεθος μακρομπλοκ, το τρέχον πλαίσιο, το πλαίσιο αναφοράς και το είδος της αναζήτησης (εξαντλητική ή λογαριθμική). Αρχικά αρχικοποιείται ο δισδιάστατος πίνακας motion_vectors με 0 με μέγεθος ύψος/(μέγεθος μακρομπλοκ) και πλάτος/(μέγεθος μακρομπλοκ). Είναι δισδιάστατος αφού θα αποθηκεύει διανύσματα μορφής (dx, dy). Επιπλέον αρχικοποιείται με 0 ο πίνακας errors στον οποίο θα αποθηκεύονται οι εικόνες σφάλματος, σε διαστάσεις ίδιες με αυτές του τρέχοντος πλαισίου. Η αρχικοποίηση έγινε καθώς διευκολύνει στην συνέχεια την ροή του αλγόριθμου, όπου για να προστίθεται κάθε φορά ένα καινούργιο διάνυσμα ή ένα καινούργιο πλαίσιο διαφορών πρέπει απλώς να ενημερώνεται η αντίστοιχη θέση του αντίστοιχου πίνακα. Για την εξαγωγή των πλαισίων διαφορών και των διανυσμάτων κίνησης ακολουθείται η εξής διαδικασία:

1. Υπάρχει ένας 3πλος βρόχος, ο οποίος πραγματοποιείται για τις τιμές του y (ύψος πλαισίου), του x (πλάτος πλαισίου) και των καναλιών χρώματος (αριθμός χρωμάτων πλαισίου, π.χ. RGB τότε 3 χρωματικά κανάλια). Το τελευταίο συμβαίνει ώστε να απλοποιηθεί η διαδικασία και κάθε φορά να λαμβάνεται υπόψιν μία διάσταση.

a. Εξάγεται το τρέχον μακρομπλοκ από το τρέχον πλαίσιο δηλαδή από την θέση *(y)* εώς *y + macroblock_size* και *x* εώς *x + macroblock_size* αντίστοιχα.
b. Ελέγχεται ποια μέθοδος αναζήτησης καλέστηκε, εξαντλητική ή λογαριθμική
c. Εξάγονται το καλύτερο ταίριασμα καθώς και τα αντίστοιχα διανύσματα κίνησης
d. Ενημερώνονται κατάλληλα οι αντίστοιχες θέσεις των πινάκων errors και motion_vectors

Η συνάρτηση επιστρέφει τους πίνακες με τα διανύσματα κίνησης και τα πλαίσια διαφορών.

2.3.1.5 ii_encoder.py

Το αρχείο αυτό περιέχει τον κωδικοποιητή για το ερώτημα ii. Ακολουθώντας όμοια διαδικασία με το αυτή στο *i_encoder.py,* ορίζεται η έξοδος (βίντεο) για το κωδικοποιημένο βίντεο *encoded_ii.avi* καθώς και η έξοδος (βίντεο) για τα πλαίσια διαφορών *differences_ii.avi.* Επιπλέον, αυτή την φορά θα υπάρχουν δύο binary αρχεία, το ένα θα αποθηκεύει τα κωδικοποιημένα πλαίσια διαφορών (που προέκυψαν βάση της εξαντλητικής αναζήτησης) και τα πλαίσια I, όταν αυτά συναντώνται στο *error_frames.bin* και το άλλο θα αποθηκεύει τα κωδικοποιημένα διανύσματα κίνησης στο *motion_vectors.bin.* Χρησιμοποιήθηκε ακτίνα αναζήτησης k=8 και μέγεθος_μακρομπλοκ=16x16, ενώ καλέστηκε η συνάρτηση motion_compensation() για την εξαντλητική αναζήτηση. Επιπροσθέτως, οπτικοποιούνται τα διανύσματα κίνησης πριν γραφτούν στην έξοδο με χρήση της συνάρτησης *arrowedLine().* Να σημειωθεί ότι **η συνάρτηση δεν συνεισφέρει καθόλου στον συνολικό αλγόριθμο δηλαδή στον υπολογισμό των διανυσμάτων κίνησης** αλλά υπάρχει μόνο για οπτικοποιήση και μάλιστα μπορεί να παραληφθεί χωρίς να επηρεάσει καθόλου τη λειτουργία του κωδικοποιητή.
Συνεχίζοντας τα πλαίσια I και τα πλαίσια διαφορών κωδικοποιούνται μέσω Huffman σύμφωνα με την ίδια διαδικασία που παρουσιάστηκε στο *i_encoder.py,* δημιουργώντας επομένως τα αρχεία *motion_vectors_codes.pkl* και *error_frames_codes.pkl* .

2.3.1.6 ii_decoder.py

Το αρχείο αυτό περιέχει τον αποκωδικοποιητή για το ερώτημα ii. Ακολουθώντας όμοια διαδικασία με το αυτή στο *i_decoder.py,* ορίζεται η έξοδος (βίντεο) για το αποκωδικοποιημένο βίντεο *decoded_ii.avi.* Αφού αποκωδικοποιηθούν τα αρχεία *error_frames.bin* και *motion_vectors.bin* σύμφωνα με τα αρχεία *motion_vectors_codes.pkl* και *error_frames_codes.pkl,* εάν το τρέχον πλαίσιο είναι P τότε αυτό θα πρέπει να ανακατασκευαστεί σύμφωνα με τα διανύσματα κίνησης και να προστεθεί στο πλαίσιο διαφορών. Αυτό πραγματοποιείται εάν ακολουθήσουμε την εξής διαδικασία:

1. Αρχικοποιείται ένας πίνακας με 0, σε διαστάσεις του τρέχοντος πλαισίου ώστε να ενημερώνεται η αντίστοιχη θέση του με την νέα τιμή, όταν ανακατασκευάζεται, με όνομα reconstructed_frame.
2. Μέσω ενός βρόχου εξάγεται κάθε φορά το διάνυσμα κίνησης από τον πίνακα motion_vectors
3. Υπολογίζουμε το αρχικό και το τελικό σημείο με βάση το μέγεθος του μακρομπλοκ *(x*μεγεθος μακρομπλοκ, y*μεγεθος μακρομπλοκ)* ως αρχικό και *(αρχικό x + dx, αρχικό y + dy)* ως τελικό.

4. Εξετάζουμε εάν το τελικά σημείο βρίσκεται εντός των ορίων του μακρομπλοκ, δηλαδή ότι δεν ξεπερνάει το ύψος και το πλάτος του.
5. Εξετάζουμε εάν το μακρομπλοκ βρίσκεται εντός των ορίων του πλαισίου, δηλαδή ότι δεν ξεπερνάει το ύψος και το πλάτος του.
6. Στη συνέχεια ανακατασκευάζεται το πλαίσιο, εξάγοντας τα αντίστοιχα μακρομπλοκ από της αντίστοιχες θέσεις του προηγούμενου πλαισίου και ευημερώντας τον πίνακα reconstructed_frame στην αντίστοιχη θέση για κάθε χρωματικό κανάλι.

2.3.1.7 iii_encoder.py

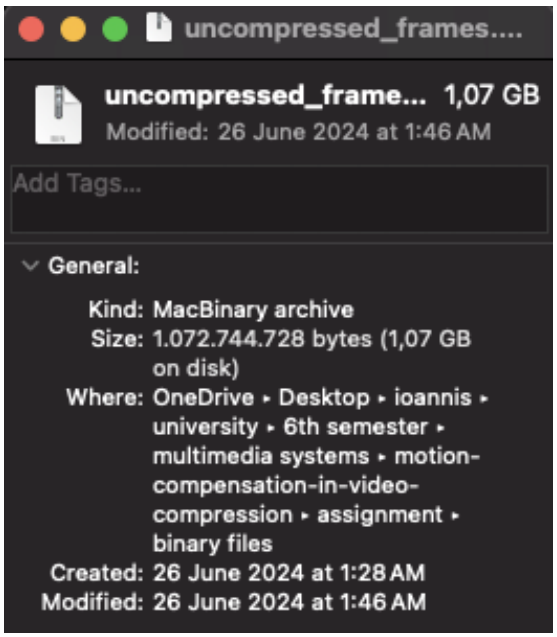
Το αρχείο αυτό περιέχει τον κωδικοποιητή για το ερώτημα ii. Ακολουθώντας όμοια διαδικασία με το αυτή στο *i_encoder.py,* ορίζεται η έξοδος (βίντεο) για το κωδικοποιημένο βίντεο *encoded_iii.avi* καθώς και η έξοδος (βίντεο) για τα πλαίσια διαφορών *differences_iii.avi.* Επιπλέον, αυτή την φορά θα υπάρχουν δύο binary αρχεία, το ένα θα αποθηκεύει τα κωδικοποιημένα πλαίσια διαφορών (που προέκυψαν βάση της εξαντλητικής αναζήτησης) και τα πλαίσια I, όταν αυτά συναντώνται στο *error_frames_log.bin* και το άλλο θα αποθηκεύει τα κωδικοποιημένα διανύσματα κίνησης στο *motion_vectors_log.bin.* Χρησιμοποιήθηκε ακτίνα αναζήτησης k=8 και μέγεθος_μακρομπλοκ=16x16, ενώ καλέστηκε η συνάρτηση motion_compensation() για την λογαριθμική αναζήτηση. Επιπροσθέτως, οπτικοποιούνται τα διανύσματα κίνησης πριν γραφτούν στην έξοδο με χρήση της συνάρτησης *arrowedLine().* Να σημειωθεί ότι **η συνάρτηση δεν συνεισφέρει καθόλου στον συνολικό αλγόριθμο δηλαδή στον υπολογισμό των διανυσμάτων κίνησης** αλλά υπάρχει μόνο για οπτικοποιήση και μάλιστα μπορεί να παραληφθεί χωρίς να επηρεάσει καθόλου τη λειτουργία του κωδικοποιητή.
Συνεχίζοντας τα πλαίσια I και τα πλαίσια διαφορών κωδικοποιούνται μέσω Huffman σύμφωνα με την ίδια διαδικασία που παρουσιάστηκε στο *i_encoder.py,* δημιουργώντας επομένως τα αρχεία *motion_vectors_codes_log.pkl* και *error_frames_codes_log.pkl* .

2.3.1.8 iii_decoder.py

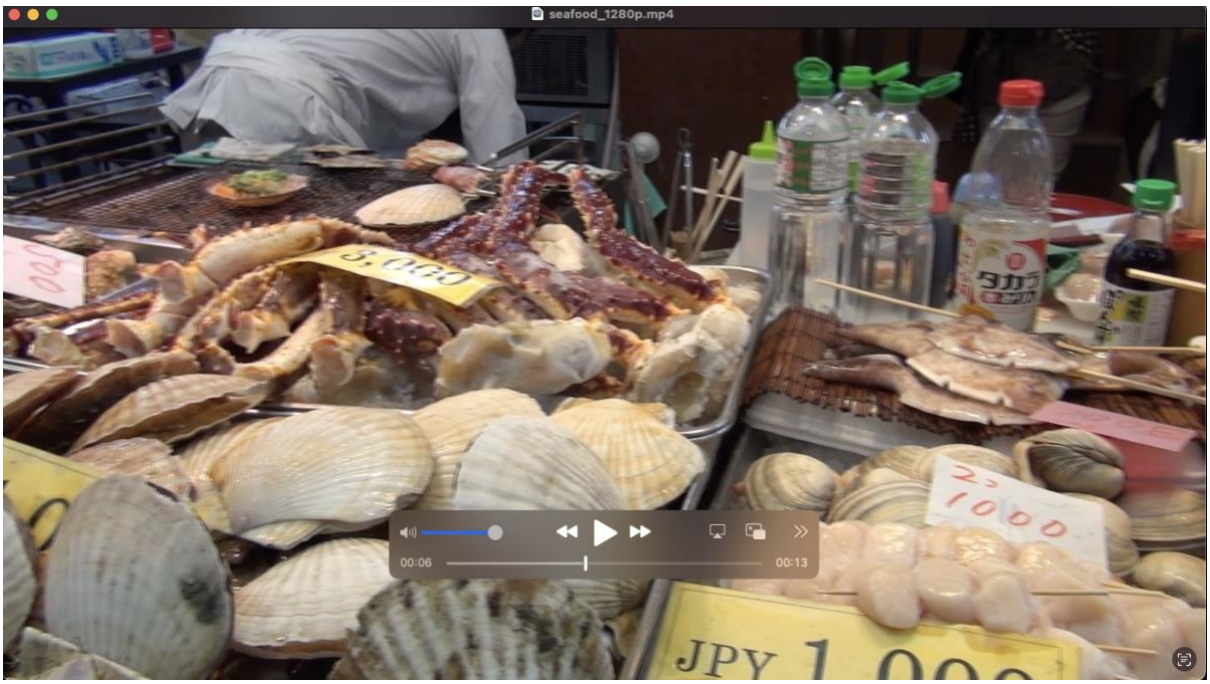
Το αρχείο αυτό περιέχει τον αποκωδικοποιητή για το ερώτημα ii. Ακολουθώντας όμοια διαδικασία με το αυτή στο *i_decoder.py,* ορίζεται η έξοδος (βίντεο) για το αποκωδικοποιημένο βίντεο *decoded_iii.avi.* Αφού αποκωδικοποιηθούν τα αρχεία *error_frames_log.bin* και *motion_vectors_log.bin* σύμφωνα με τα αρχεία *motion_vectors_codes_log.pkl* και *error_frames_codes_log.pkl,* εάν το τρέχον πλαίσιο είναι P τότε αυτό θα πρέπει να ανακατασκευαστεί σύμφωνα με τα διανύσματα κίνησης και να προστεθεί στο πλαίσιο διαφορών. Αυτό πραγματοποιείται εάν ακολουθήσουμε την εξής διαδικασία που περιγράφτηκε στο *ii_decoder.py.*

3. ΕΠΙΔΕΙΞΗ ΤΗΣ ΛΥΣΗΣ

Για να αναδείξουμε την λειτουργία των προγραμμάτων θα τρέξουν για ένα βίντεο ήπιας κίνησης, το οποίο απεικονίζει θαλασσινά σε μία λαϊκή αγορά της Ιαπωνίας. Το βίντεο έχει διάρκεια 13 δευτερόλεπτα, είναι RGB και σε διαστάσεις 720x1280, όπως αναφέρθηκε παραπάνω. Το αρχικό μέγεθος του ασυμπίεστου βίντεο αφού γραφτεί σε binary αρχείο είναι 1.07 GB, ενώ του συμπιεσμένου θα είναι 815.4 MB (δεν θα ήταν λάθος να θεωρούνται αμελητέα τα αρχεία .pkl με τους κώδικες Huffman εφόσον συντελούν ελάχιστα MB)

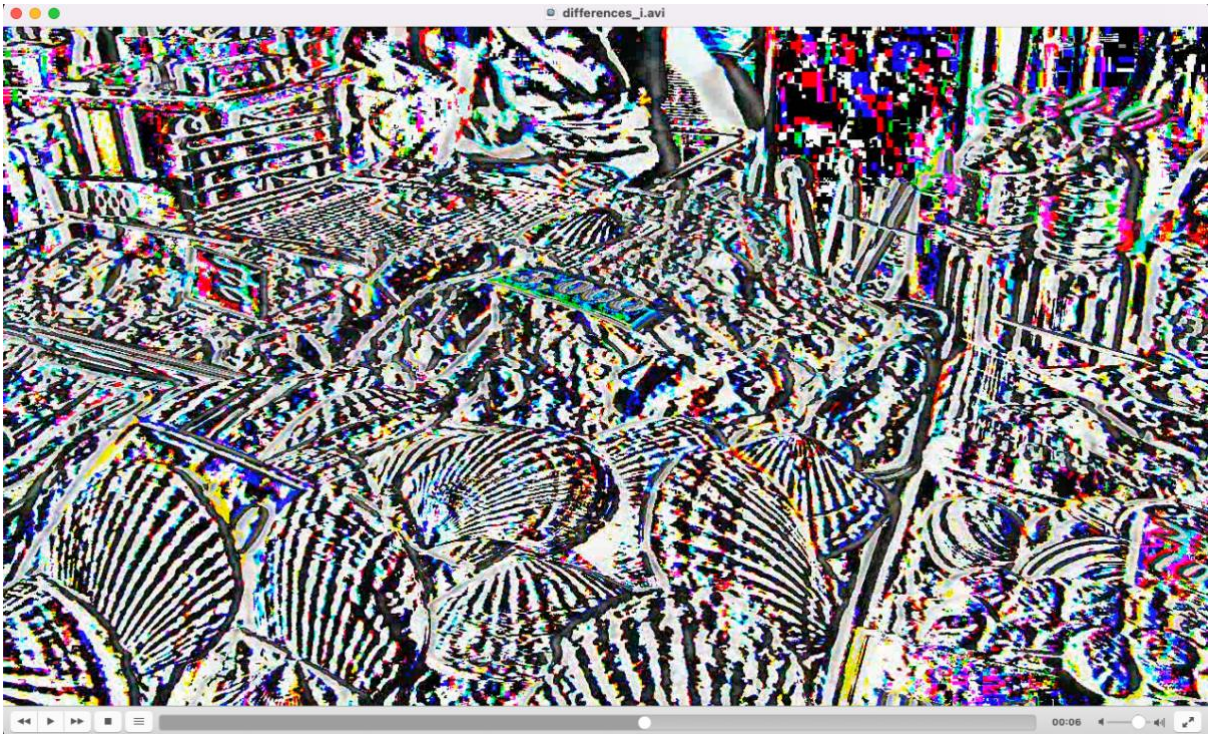


Εικόνα 1. Μέγεθος αρχείου uncompressed_frames.bin

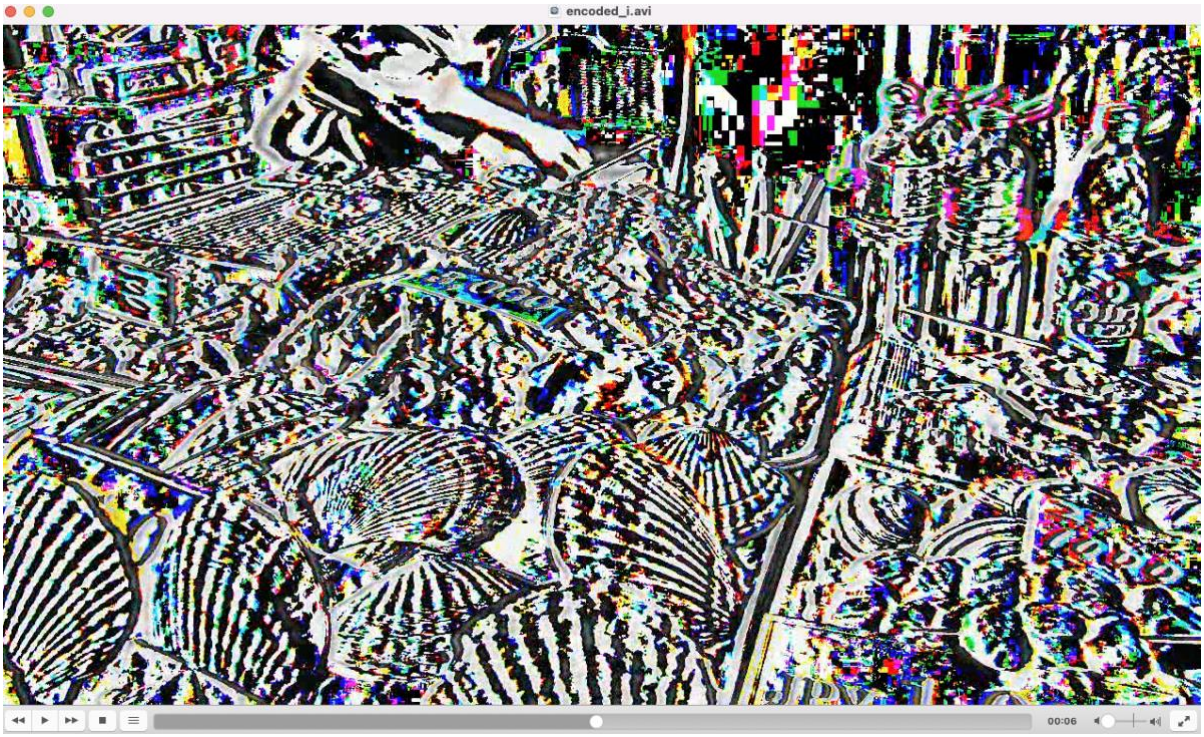


Εικόνα 2. Αρχικό βίντεο

3.1.1 i

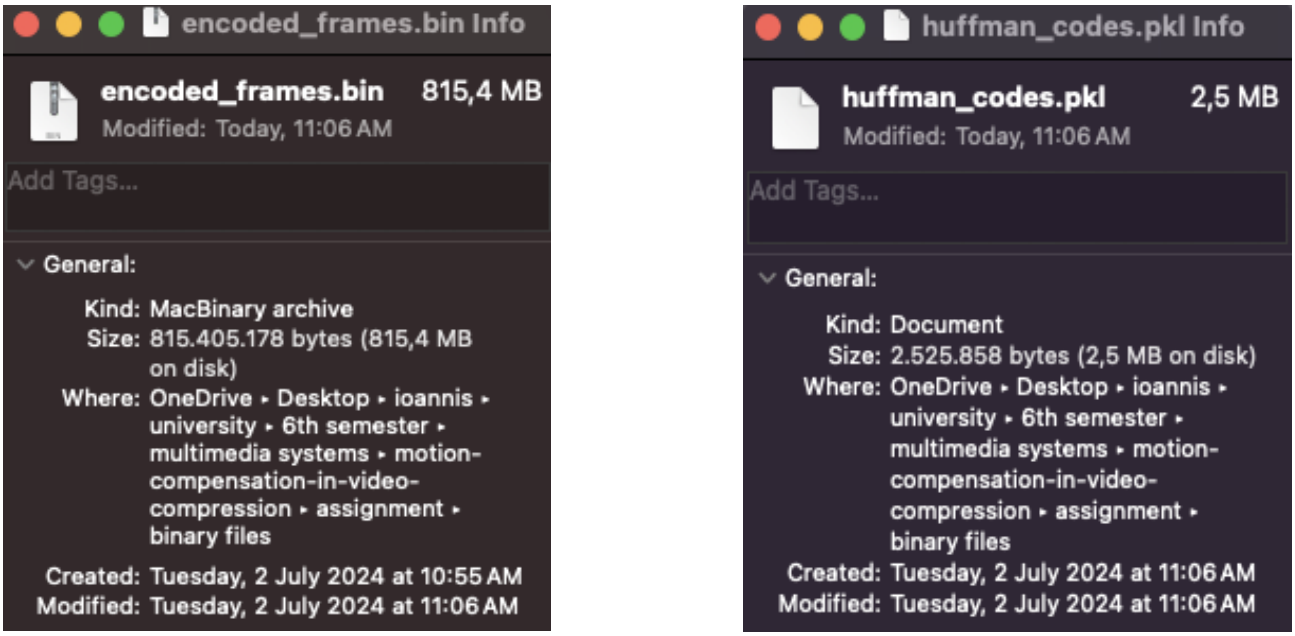


Εικόνα 3. Ακολουθία εικόνων διαφορών i



Εικόνα 4. Κωδικοποιημένο βίντεο i (ίδια με ακολουθία διαφορών με τα I πλαίσια να παρεμβάλλονται)

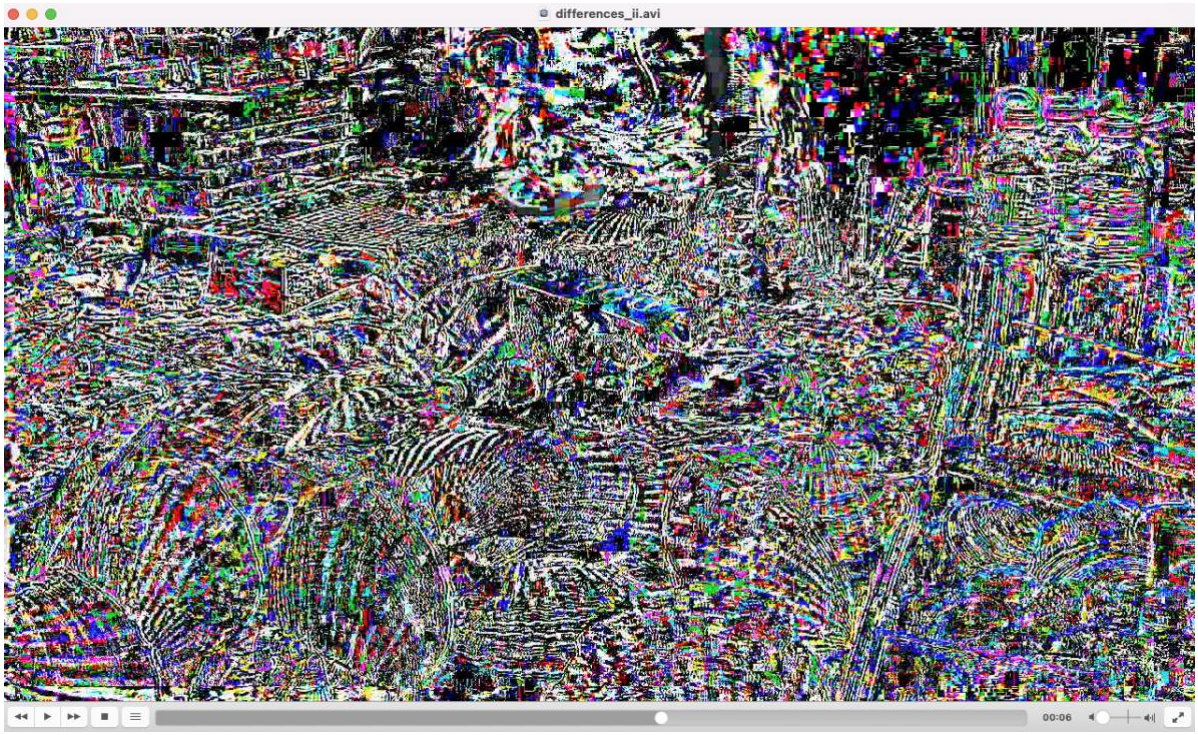
Εικόνα 5. Αποκωδικοποιημένο βίντεο i



Εικόνα 6. Μέγεθος αρχείων encoded_frames.bin και huffman_codes.pkl

Παρατηρείται πως το μέγεθος του αρχείου έχει μειωθεί κατά περίπου 20 % σε σχέση με το αρχικό.

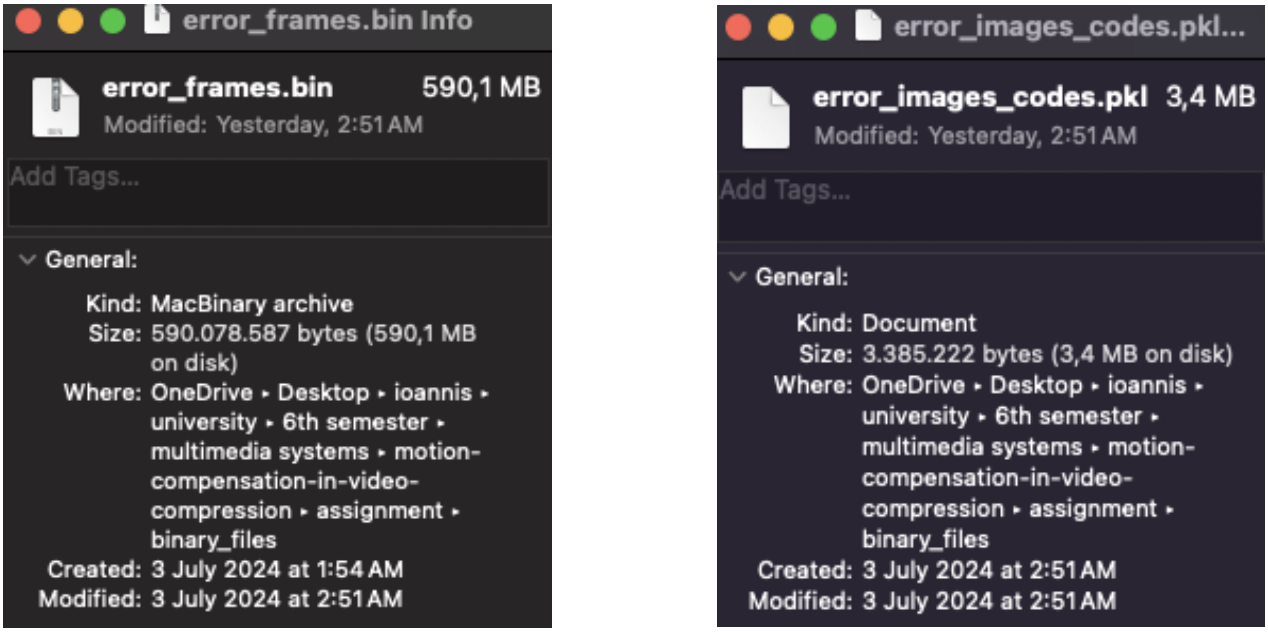
3.1.2 ii



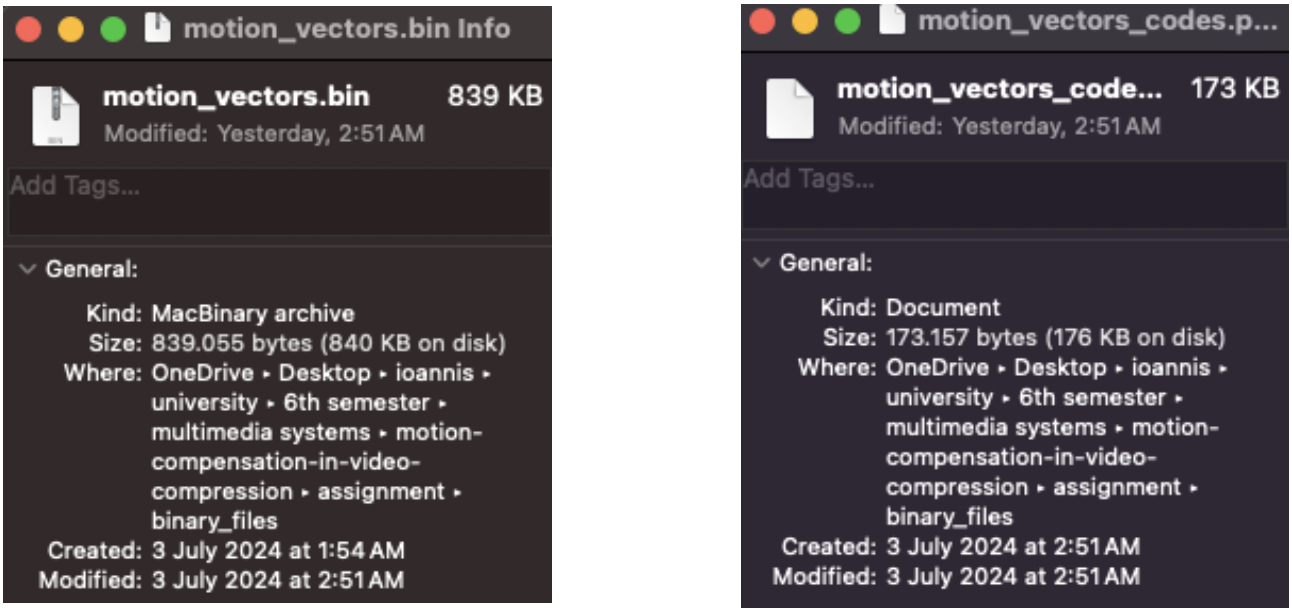
Εικόνα 7. Ακολουθία εικόνων διαφορών ii

Εικόνα 8. Κωδικοποιημένο βίντεο ii, μαζί με τα διανύσματα κίνησης

Εικόνα 9. Αποκωδικοποιημένο βίντεο ii



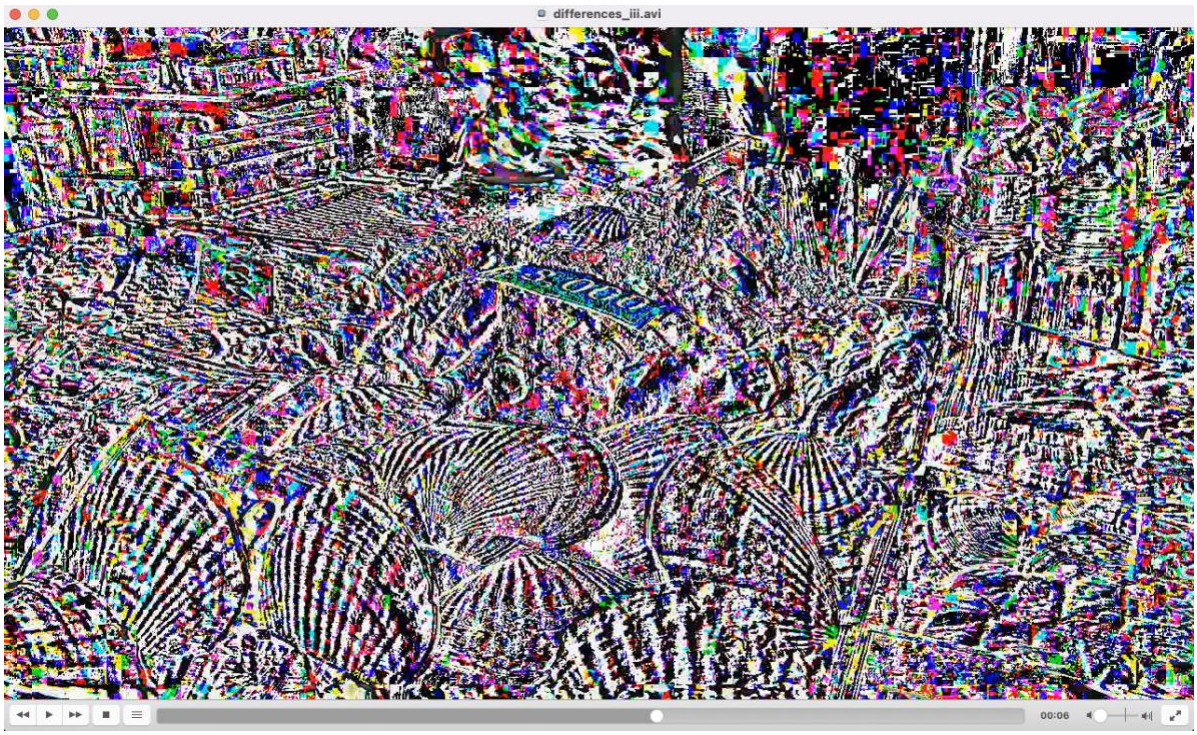
Εικόνα 10. Μέγεθος αρχείων error_frames.bin και error_images_codes.pkl



Εικόνα 11. Μέγεθος αρχείων motion_vectors.bin και motion_vectors_codes.pkl

Παρατηρείται πως το μέγεθος του αρχείου έχει μειωθεί κατά περίπου 40 % σε σχέση με το αρχικό.

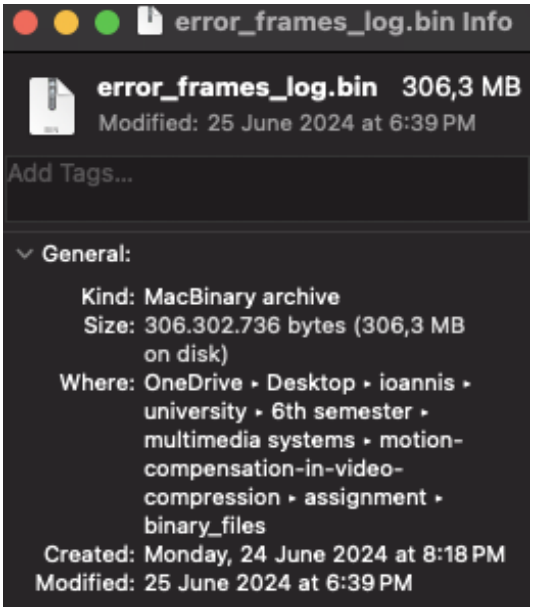
3.1.3 iii

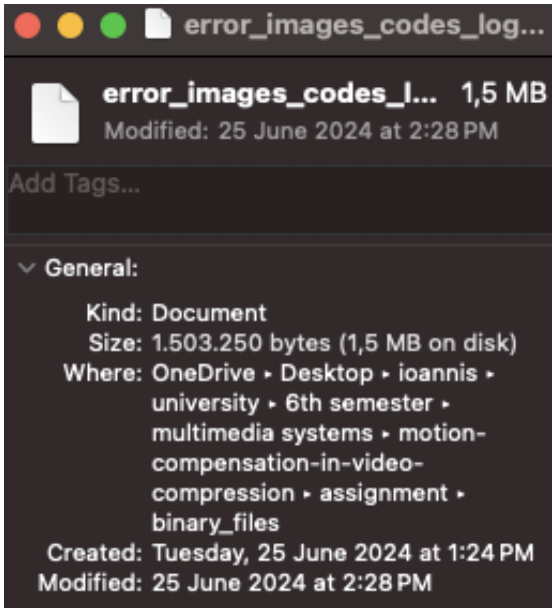


Εικόνα 12. Ακολουθία εικόνων διαφορών iii

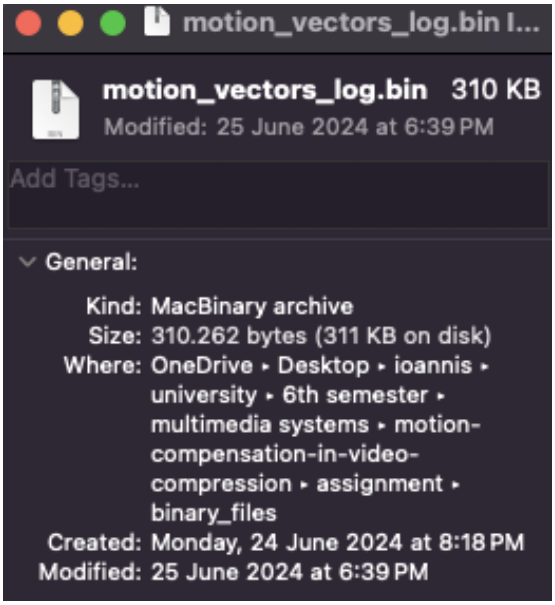
Εικόνα 13. Κωδικοποιημένο βίντεο iii, μαζί με τα διανύσματα κίνησης

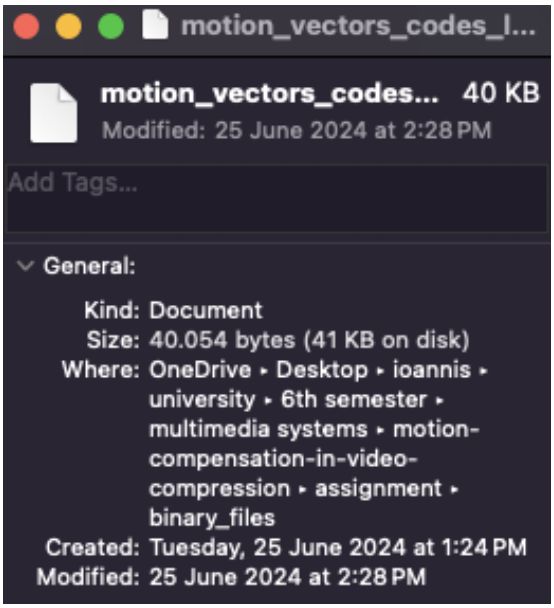
Εικόνα 14. Αποκωδικοποιημένο βίντεο iii





Εικόνα 15. Μέγεθος αρχείων encoded_frames_log.bin και huffman_codes_log.pkl





Εικόνα 16. Μέγεθος αρχείων encoded_frames_log.bin και huffman_codes_log.pkl

Παρατηρείται πως το μέγεθος του αρχείου έχει μειωθεί κατά περίπου 70 % σε σχέση με το αρχικό.

3.1.4 iv

Το μέγεθος του ασυμπίεστου βίντεο σε δυαδικό αρχείο είναι 1,072,744,728 bytes ή 8,581,957,824 bit. Το μέγεθος του αρχείου που περιέχει τα κωδικοποιημένα πλαίσια του ερωτήματος i είναι 815,405,178 bytes ή 6,523,241,424 και σε αυτό προστίθεται επιπλέον το αρχείο που περιέχει τους κωδικούς Huffman, μεγέθους 2,525,858 byte ή 20,206,864 bit, άρα συνολικά 6,543,448,288 bit. Επομένως ο λόγος συμπίεσης θα είναι 8,581,957,824 / 6,543,448,288 = 1.311534445796479 για το ερώτημα i.

Όσον αφορά το ερώτημα iii, το μέγεθος του αρχείου που περιέχει τα κωδικοποιημένα πλαίσια του ερωτήματος ii είναι 306,302,737 byte ή 2,450,421,896 bit και το αρχείο με τους κωδικούς Huffman, με μέγεθος 1.503.250 byte ή 12,026,000 bit τα οποία προστίθεται στο μέγεθος του αρχείου που περιέχει τα διανύσματα κίνησης, μεγέθους 310,262 bytes ή 2,482,096 bit και το αρχείο με που περιέχει τους κωδικούς Huffman με μέγεθος 40,054 bytes ή 320,432 bit Αν προστεθούν δίνουν ένα αποτέλεσμα 2,465,250,424 bit. Επομένως ο λόγος συμπίεσης θα είναι 8,581,957,824 / 2,465,250,424 = 3.481170813501095.

ΒΙΒΛΙΟΓΡΑΦΙΚΕΣ ΑΝΑΦΟΡΕΣ

[1] https://www.kaggle.com/datasets/mistag/short-videos (video)

[2] https://www.youtube.com/watch?v=AxIc-vGaHQ0 (YouTube tutorial on OpenCV)

[3] https://www.opencvhelp.org/tutorials/advanced/video-compression/ (OpenCV compression tutorial)

[4] https://www.geeksforgeeks.org/huffman-coding-using-priority-queue/?ref=lbp (Huffman Encoding)

[5] https://www.geeksforgeeks.org/huffman-decoding/ (Huffman Decoding)

[6] https://www.geeksforgeeks.org/sum-of-squares-of-differences-between-all-pairs-of-an-array/ (SSD)

[7] https://www.youtube.com/watch?v=dqxbBl-BGho (motion compensation technique)

[8] https://www.geeksforgeeks.org/python-opencv-cv2-arrowedline-method/ (for showing motion vectors)

[9] Σύγγραμμα μαθήματος